G.F. CHAMBERS

S I N C - L I N K $1.50 per copy

The Toronto Timex/Sinclair Users Club Newsletter Vol. 1, No.2

P.O. Box 7274, Stn. A Toronto M5W 1X9

In this Issue:

From the President:

Dear Members:

Late in 1982, 5 ZX-81 owners met in a bar at Yonge & Eglinton in Toronto to talk about their computers. These Gentlemen are now the Executive of the Toronto Timex-Sinclair users Club. We are now a National Club with members as far away as B.C. We now have 72 members. The Club meets the first and third Wednesday of every Month at the North York Community Hall at Yonge & Sheppard in Willowdale. Our last meeting before summer break will be Wednesday, June 22. Regular meetings will resume Wednesday, September 21. The Club's financial status is OK for the time being, so there will not be an admission charge per meeting, as has been suggested. I would like to remind everyone, if you have any comments, questions or suggestions of any kind, please feel free to let me know. I hope you enjoy this long awaited Newsletter.

Yours sincerely,

Pete Harvey

Pete Harvey

EXECUTIVE

Librarian - John Castillos

Asst. Librarian - Martin Mauk

New Members - George Chambers

Membership Secretary and Education - Greg Lloyd

Correspondence - Chris Hart

Newsletter prepared by GRAPHIC ENTERPRISES, WORD PROCESSING CENTRE

HARDWARE REVIEW
- by J.J. Castillos

MULTIFORTH EPROM CHIP	BY	THREE SYSTEMS OF GRAND RAPIDS	FOR THE ZX81

FORTH is a high-level computer language developed by Charles H. Moore in the last sixties. It was interpretive in the beginning but compiler versions are currently available among which is listed the one we are reviewing now. The basic unit in FORTH is the word, the language is usually provided with a certain number of words, each one of which defines a given function or routine and the user can add to this vocabulary with other words which he designs. Because of the compiler nature of these versions of the language, it runs very fast as compared to BASIC. Speed is not the only advantage to the user since long BASIC programs involving complicated subroutines have to be defined only once and then a short string of FORTH words can replace what would be many lines of BASIC Program.

Last year I bought the IPS version of compiler FORTH which is supplied as software in a cassette and takes about 7K of memory. When the Tree Systems version which is supplied as an 8K EPROM chip to replace the BASIC ROM became available, I decided to buy it in order to compare them.

One of the first conclusions I drew is that although both versions are roughly comparable in memory size, capabilities, speed and number of basic words provided, the Tree Systems chip has some advantages such as the extra memory available to the user. If you have a 16K RAM pack attached to your ZX81, the IPS FORTH leaves you about 9K of free RAM while the Tree Systems chip takes less than 2K for its internal use and leaves you over 14K for your programs. Other advantages are that the Tree Systems chip has a built-in editor with split screen, it has auto-repeat on all keys, it is capable of multi-tasking (ie. up to ten programs can be scheduled to run at some time in the future, for example, take temperature readings every 20 minutes, etc.), it has standard ASCII characters which are displayed on the screen and last but not least, the convenience of the chip itself which requires no loading and provides instant availability on power-up. The 2764 EPROM chip containing FORTH comes soldered to a small board to which the BASIC ROM can also be attached so as to enable the user to switch at will between BASIC and FORTH.

One serious drawback common to both versions currently available for the ZX81 is the inability to obtain printouts, the results are just displayed on the screen. Also the claims by Tree Systems that their version of FORTH runs up to 130 times faster than BASIC is wildly exaggerated and misleading. I tried the following standard test in both BASIC and FORTH:

BASIC	FORTH
10 For A = 1 to 3000 20 Scroll 30 Print A 40 Next A	3000 1 DO CR I . LOOP

In both FAST and SLOW modes FORTH was between 5 and 7 times faster than BASIC which is a significant advantage but nothing even close to the above claim.

The FORTH chip plus board comes with a 110 page manual which although full of irritating spelling errors, is well written and to the best of our knowledge, quite error-free, a refreshing change from IPS's notoriously defective manual.

I must warn potential users that the Tree Systems version of FORTH requires at least 2K of RAM and also that the power supply used will have to provide more than 500 mA. The BASIC computer plus 16K RAM which we used for the test worked well with such a power supply but with the FORTH chip plugged in the same system was useless due to screen instability. A 650 mA power supply solved the problem. Finally, you must remember to turn off the computer before attempting to switch from one ROM to the other.

Summing up, if you want to do serious programming in FORTH, the Tree Systems version is good value for your money unless you wish to wait to see if a chip becomes available in the future to provide access to the ZX Printer and perhaps more fig-FORTH commands. On the other hand, if you only want to see what FORTH is like and just try it out, then the IPS software version (ZX FORTH) should suffice.

The FORTH EPROM chip plus board and manual is available from Tree Systems of Grand Rapids, Michigan, USA for US$ 49.95.

B O O K   R E V I E W
by J. J. Castillos

MASTERING YOUR TIMEX SINCLAIR 1000 PERSONAL COMPUTER by Tim Hartnell and Dilwyn Jones, BANTAM BOOKS paperback, New york/Toronto, $3.95 (COLES BOOKS).

At last a comprehensive book to introduce users to ZX81 BASIC at a reasonable price! A puzzling fact that caused me great concern when I began buying computer books was that the prices were remarkably high for shoddily produced, thin paperbacks, often full of frustrating errors both in the text and in the program listings. Some people were obviously making huge profits with very little effort at the expense of us users who had at the time very little choice.

The above book written by Tim Hartnell, former Editor of ZX COMPUTING magazine, and Dilwyn Jones, a user himself and founder of a ZX81 Club in North Wales, is one of the best introductions to Sinclair BASIC that I have seen. The book starts assuming that you know absolutely nothing about computers and devotes several chapters to explain the different commands available in Sinclair BASIC. A study of conditional statements, loops and arrays follows before the authors wrap up their description with moving graphics and the more esoteric PEEKs and POKEs. The book ends with two valuable chapters on Business Applications and on converting other BASIC listings to the ZX81 version. The latter is the most complete list of functions and their Sinclair equivalents that I have seen so far, 22 pages in all. Even somewhat experienced people such as myself gained valuable new knowledge after reading this book which includes dozens of examples to illustrate every point the authors make. For instance, I knew that by POKEing 16418,0 I could PRINT on all 24 lines of the screen but I did not know that by POKEing numbers between 3 and 21 in that location, a SCROLL will begin above the normal starting point (line 21) leaving variable portions of the screen untouched. Try the following: 10 PRINT AT 17,0; "UNAFFECTED LINES" 20 PRINT AT 21,0; "OF THE SCREEN..." 30 POKE 16418,10 40 SCROLL 50 PRINT "A LINE OF TEXT TO BE SCROLLED UP" 60 GOTO 40. RUN this program in SLOW mode and see what happens.

The book is well written, in plain English and with a no-nonsense approach, and will be very useful to any serious ZX81 user. I tried hard to find errors or obscure points but could not find any, which is a recommendation in itself.

===

A SHORT PROGRAM IN FORTH (ZX FORTH)
by - J. J. Castillos

If you are like me, you got your copy of ZX FORTH, loaded it, played a bit with it and after realizing how different it was from BASIC, you put it away for a while. Just now I am beginning to grasp the elementary notions of FORTH programming and I would like to share with you a short program which shows some simple graphic and printing routines (each # represents one blank space). Compare the speed of execution with a similar BASIC program, for example, any of the BASIC games you may have which starts by drawing a black border around the screen.

```
                                   (30 spaces)
:  G 24 1 DO CR ."##" 128 EMIT ."##############################" 128 EMIT LOOP ; Newline

:  A ."##" 31 Ø DO 128 EMIT LOOP ;  Newline

:  M 10 1 DO CR LOOP ;  Newline

:  E ."##" 128 EMIT ."####THIS IS A DEMONSTRATION" CR CR ;  Newline

:  S ."##" 128 EMIT ."#####OF FORTH GRAPHICS..." CR CR ;  Newline

                       (14 spaces)
:  X ."##" 128 EMIT ."##############" ;  Newline
```

continued....

After making the above definitions exactly as listed, you can run the program by entering the following list of words:

TASK G A HOME A M E S X Newline

and see what happens. Remember that after the laborious entering of the definitions, these stay in memory and are added to your Vocabulary, the actual program is the last short line of words.

==

BUG-BYTE ZXTK Application Notes

- by G.F. Chambers

As many purchasers of the Bugbyte ZXTK Tool kit program are no doubt aware the instruction leaflet supplied with the cassette can be rather daunting. For the benefit of the neophyte programmer like myself, I have put together these notes which should help in getting started. Note these are applicable to the ZX81 with a Sinclair 16K RAM attached.

STEP	ACTION	ON-SCREEN RESULT
1	Load ZXTK (Tool kit)	On completion, displays advice to stop; after a pause the option list appears.
2	Stop tape recorder	---
3	Press BREAK key	Report code (inverse D/0) appears.
4	Enter POKE 16389, 120	Screen clears and Report code (0/0) appears.
5	Enter RAND USR 17034	Menu reappears.
6	Press 8	Fig 8 appears as OPT=8.
7	Press N/L	Start = appears.
8	Press 1	Fig 1 appears as START = 1.
9	Press N/L	END = appears.
10	Press 3	Fig 3 appears as END=3.
11	Press N/L	Inverse 0 appears.
12	Press BREAK	Report code (inverse D/0) appears.

This sequence has placed ZXTK safely above RAMTOP. Now proceed to load a program in the normal fashion (step 13).

STEP	ACTION	ON-SCREEN RESULT
13	Load your program	----
14	Enter PRINT PEEK 31954	205 appears in upper RH corner.
15	Enter RAND USR 31954	Report code (0/0) appears.
16	Enter RAND USR 17034	Menu re-appears.

You are now ready to apply the ZXTK functions to your program. As an example, to renumber a program to start at line 100 and increment in steps of 10 line numbers, follow the sequence below:

STEP	ACTION	ON-SCREEN RESULT
1	PRESS 2	OPT=2
2	PRESS N/L	START =
3	PRESS 100	start = 100

continued.....

STEP	ACTION	ON-SCREEN RESULT
4	PRESS N/L	Step =
5	PRESS 10	Step = 10
6	PRESS N/L	Screen goes blank for several seconds. Display reappears with a ZXTK successful completion report code (Inverse 0).

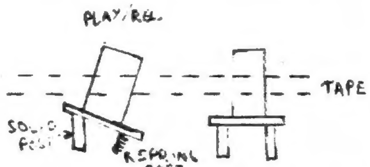
Your program is now renumbered. You may observe this by entering BREAK, then LIST 100. Enter RAND USE 17034 to get back to ZXTK menu.

TAPE HEAD ALIGNMENT

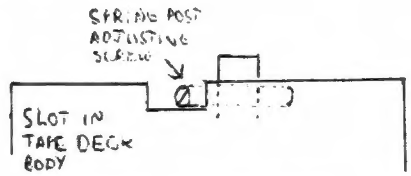
- by Stan Piotrowski

If you are experiencing LOADing problems with commercial or tapes from our club library, in all probability your "Azimuth" is not correct. (Fig.1). This simply means that the tape is not running at right angles to the tape head. The following tests and corrections should be made to avoid frustration both to yourself and our librarian. The best time is when you first obtain a new or different tape recorder in order to avoid re-SAVEing all your programs after your recorder is set perfectly.

FIG 1



FIG



Unplug the tape deck and without any tape in it, press the "Play" button. There are two tape heads that will move forward (your erase and Record/Playback heads). On closer inspection, you will see two solid supporting posts under one head. This is the Erase Head. Under the other head you will see one solid supporting post and one spring-type post. This is the head you will be adjusting. Now, take note that above this spring post is a slotted screw. This is where you will make your adjustments for perfect LOADing. Above this screw, there is usually a slot in the body of the recorder in order to fit a small screw driver so adjustments can be made with the tape running. (Fig.2).

Press the "Stop" button so the tape heads move back to their resting or "not-playing" position. Plug the tape recorder in and use either a commercial ZX-81/Timex 1000 computer tape or one from the library. (The tape recorder should not be hooked up to your computer at this time). Turn the volume down so you can tolerate the whining noise and can clearly hear the differences in pitch and clarity of the tape when you make your adjustments.

Now, slowly turn the screw first in one direction then the other and you will immediately hear the sound becoming more BASS or deeper in tone. Turn the screw head back until you can hear the sound at its highest treble or clearest pitch. This is the ideal setting and means you have corrected the "Azimuth" problem. The screw should not have turned anymore than 1/2 turn from its' original position. Remember, DO NOT USE ONE OF YOUR TAPES for this test.

Now that you have your tape deck all tuned up correctly, you may not be able to load all your programs since they were SAVEd when the heads were mis-aligned. There are two alternatives: 1. use another tape deck that has tape recorder (before you align the head).
2. a slower but effective means using your tape deck alone:-

To use your tape deck alone you will have to constantly adjust the "Azimuth" adjusting screw by this method: keep that library tape or commercial tape handy. Put your tape back into the player and adjust the screw until you hear the highest treble. Adjust the volume to your normal LOAD level, plug the tape into the computer and LOAD the program. Remove your tape and place the Library or Commercial tape back in and with the tape recorder unplugged from the computer, adjust the screw until you again hear the highest treble. SAVE your program as per normal volume etc.

This is the only solution to your problem, but once completed, your programs should LOAD properly. If you are lucky, you have only a few programs to LOAD & SAVE by this method, otherwise like me, a lot of hours will be spent in correcting all your previous SAVEd programs.

UNDERSTANDING AND USING PEEK AND POKE

- by Stan Plotrowski

All the memory used by computers is composed of numbers which are called addresses which is the same as line numbers in Basic. In the ROM memory, the addresses used by the computer are from 0 to about 8100. From 8192 to 16383 is the "Ghost area" of the ZX-81 where there is no memory. From 16383 to 16513 is the memory used by the computer to keep track of how much memory your computer has, or how many lines you are able to use on the screen, etc. (your manual gives specific uses of these addresses). From 16514 to 32768 (for 16K RAM packs) is the available memory to be used by you in your programs. The ROM memory addreses are "burned" in with their specific numbers that tell the computer how to print the letter "A" for example and as such cannot be changed by you. The system variables that uses addresses 16383 to 16513 can, in some instances, be changed by you without any crashes.

```
10  PRINT PEEK 16388
20  PRINT PEEP 16389
    RUN & ENTER
```

The first number on the screen will be Ø and the second number will be 128. Without going into full detail on machine code, in some instances the addresses hold an arithmetic number (as opposed to a number that is a machine code command). The peculiar thing about 2 addresses that have numbers to be used in arithmetic form, is the way they hold them. The greatest number that any address can hold is 255 and the lowest is 0. Therefore, any address can hold one number from Ø to 255 or 256 possible numbers (0 is a number). 256 is not very high so the computer sientist enable us to count higher by using the combination of 2 addresses - the first address would contain a number from 0 to 256 and likewise the second address. But, the second address with a number would be multiplied by 256 and added to the first number. In the above program 16388 contains 0 (the first address contains the "Low" byte) and the second address (the "High" byte) contains 128. Therefore, 256*128 = 32768 + 0 = 32768. This is the last address for RAMTOP. In some programs you are asked to change Ramtop by lowering it so that you can place machine code routines above it. These are the addresses used by the computer to keep track of Ramtop so by POKEing different values into these addresses you "fool" the computer into thinking there is only room in memory from 16514 to the address you POKEd, for Basic. (Only machine code can be entered above Ramtop). Therefore, if you want to reserve 1000 Bytes above Ramtop for machine code routines you can work it out mathematically:

1. 32768 - 1000 = 31768	:obtain value of new Ramtop
2. INT (31768/256) = 124	:value for address 16389 (remember High byte second, low byte first)
3. 124*256 = 31744	
4. 31768 - 31744 = 24	:value for address 16388

Now, by direct command, POKE 16388,24
POKE 16389,124
New
ENTER

You have now reserved 1000 bytes above Ramtop for your machine code routines. To verify this: PRINT PEEK 16388 + 256 * PEEK 16389 and your answer will be 31768. As far as the computer is concerned the top of available memory is 31768.

Turn the computer off then on again and by direct command: PRINT PEEK 16388 - 256 * PEEK 16389 and the computer will now tell you the end of available memory (Ramtop) is at address 32768. By the way, 32768 + 16514 = approximately 16000 which is 16K RAM.

If you noticed in some commercial programs, the first line number (which may contain machine code) is the number "0". In Basic you cannot enter this as a line number nor can you delete a line that is 0. You can alter the first line number or make your first line number a 0 by POKEing an address:

```
1  REM  ZX-81 COMPUTER
```

BY DIRECT COMMAND: POKE 16510,0 then LIST 0. You will see that the first line number in your program is now a zero. If you POKE 16510,10 your first line number will be 10.

continued........

Another system variable address you can change with discretion is 16418. This address holds the number of lines you are able to print on. PRINT PEEK 16418 and your answer will be 2. When the computer is first turned on, it places 2 at the above location which allows you to print or use line numbers 0 to 21 but you can use the full 24 lines (0 to 23) of screen. The number 2 is there so that there is room for scroll and INPUT commands and for space to give error code reports when your program stops, such as 9/100 which means STOP statement executed at line number 100. Whenever you think of the lines on the screen such as PRINT AT 10,0 you know that the computer will PRINT something about the centre of the screen counting down from the top of the screen: 0, 1, 2, etc. The number at address 16418 counts from the bottom where in this case it means you cannot print on the bottom 3 lines: 0, 1, 2. You can change this number anywhere from 0 to 24. e.g.

```
10  POKE 16418, 10
20  PRINT AT 18,6; "ZX-81 COMPUTER"
30  SCROLL
40  PRINT TAB 10; "HELLO"
50  GOTO 30
    RUN and ENTER
```

Note where the SCROLLing begins.

To use the full 24 line screen yur program cannot contain a SCROLL or INPUT command or the program may crash.

```
10  POKE 16418,0
20  CLS
30  FOR I = 0 to 23
40  PRINT "32 inverse spaces"
50  PRINT AT I, (RND*31); "inverse period"
60  NEXT I
70  PRINT AT 10,14; "inverse<, inverse +, inverse>"
80  IF INKEY$ = "s" THEN GOTO 20
90  GOTO 80
    RUN and ENTER
```

(You need more than 1K of RAM for this program).

Press "S" and the programs starts again.

So you can see that by POKEing different values into an address you can alter something (in the above cases they were all arithmetic values) and if you learn Machine Code Language you can alter commands or anything else (or you can cause the computer to crash if you are not careful).

In some programs you may see machine code routines which can enhance or make a program run faster. Typically, a machine code program can run from 20 to 50 times as fast. For example;

10 LET A=10 : In this basic program line, the number of bytes used is 18.

- 4 for the line number
- 1 for LET
- 1 for A
- 1 for =
- 10 for 10 (5 bytes per number)
- 1 for the end-of-line

M/C: LD A, 10 - uses 2 bytes. (This means Load A with 10 which is identical to the above Basic line).

Not only does machine code use less bytes but because you are working directly with the CPU your program does not have to be interpreted from Basic to machine code and back to basic then check for syntax errors all of which are time consuming.

continued.....

The following are 2 machine code loader programs that you can use to POKE machine code into a REM statement or above RAMTOP.

Decimal

```
10  LET X=16514      :If above RAMTOP then change this value to the first address above
                     RAMTOP value.
20  INPUT Y
30  POKE X, Y
40  SCROLL
50  PRINT X, Y
60  LET X=X+1
70  GOTO 20
```

ENTERING HEX NUMBERS

```
10  LET X=16514
20  INPUT Y$
30  POKE 16*CODE Y$ + CODE Y$(2) - 476
40  SCROLL
50  PRINT 16* CODE Y$ + CODE Y$ - 476
60  LET X = X+1
70  GOTO 20
```

These loader programs enable us to quickly POKE machine code routines instead of laboriously POKEing them separately such as:

```
POKE 16514,23
POKE 16515,82
POKE 16516,20
etc.
```

To see the difference in speed in Basic compared to Machine Code try these 2 routines.

BASIC	MACHINE CODE	
10 PRINT "D" 20 GOTO 10	1 REM 1234567	POKE 16514,62 POKE 16515,53 POKE 16516,205 POKE 16517,8 POKE 16518,8 POKE 16519,24 POKE 16520,249

The following are short machine code routines placed as the first statement in your program.

All machine code routines in the following example are placed in REM statements.

continued.....

Inverse of all characters on screen:

1 REM 12345678901234567890 (20 bytes placed in REM statement).

:enter the decimal loader with X=16514 then INPUT the following numbers;

42,12,64,6,23,43,35,126,254,118,32,3,16,248,201,198,128,119,24,242

(of course you do not enter the commas)

Delete the above loader program BUT NOT THE REM STATEMENT and enter the following example program:

```
10  PRINT AT 6,8; "ZX-81 COMPUTER"; AT 9,8; "INVERSE DEMO"
20  PAUSE 50
30  RAND USR 16514
40  PAUSE 50
50  RAND USR 16514
60  STOP
```

(RAND USR is the same as RUN AND ENTER in Basic).

Delete Lines 20, 30, 40, 50 and enter the following Lines:

```
20  FOR I = 1 TO 10
30 RAND USR 16514
40 NEXT I
```

You can see where you might be able to enhance a program that you made up by adding a program line of RAND USR 16514 to get the inverse and then to change it back use of the USR command again.

SCROLL ROUTINES

You will have noticed by now how slow the SCROLL command on the ZX-81 is and the longer the program, the longer it takes to print on the screen. Also, after you have printed and scrolled numerous times, it seems to take forever to clear the screen. In the ZX-81 there is no Basic command to Scroll down. The following are machine code routines for Scrolling up or down.

1 REM (46 bytes such as 46 x X's.

Enter the Decimal Loader Program and INPUT the following numbers:

42, 12, 64, 229, 17, 33, 0, 25, 209, 1, 214, 2, 237, 176, 201, 42, 16, 64, 17, 67, 0, 237, 82, 229, 17, 33, 0, 237, 82, 209, 1, 181, 2, 237, 184, 42, 12, 64, 6, 32, 35, 54, 0, 16, 251, 201.

After entering the above numbers, delete the loader program leaving the REM statement.

Here is a short Basic program to use the scroll routines:

```
10  PRINT AT 10,10; "ZX-81 COMPUTER"
20  IF INKEY$="U" THEN RAND USR 16514
30  IF INKEY$="D" THEN RAND USE 16529
40  GOTO 20
    RUN & ENTER
```

When you press U the Print statement will move Up and by press D the statement will move down.

To make further use of the Scroll routines in a program you could do the following:

Delete all the lines except your REM statement containing the machine code for the Scroll routines. Enter the following short Basic Program.

continued.......

```
10  LET A=16514
20  LET B=16529
30  LET X=20
40  LET Y=0
```

:the letters are assigned in order to save bytes in memory since a letter takes only 1 byte whereas each number takes 5 bytes; multiply that by the number of times you will be using the routines in a program and it will be a considerable saving in bytes.

```
50  PRINT AT X,Y; "THIS IS A DEMONSTRATION IN"
60  RAND USR A
70  RAND USR A
80  FOR I = 1 TO 50
90  NEXT I
100 PRINT AT X, Y; "THE USE OF THE SCROLL ROUTINE"
110 FOR I = 1 TO 10
120 RAND USR A
130 NEXT I
140 STOP
```

Note the use of the PRINT AT statement. In the Basic Sinclair Scroll the statement will automotically print at the bottom of the screen but in machine code Scroll you must specify this.

Note also, lines 80 and 90. You could have used PAUSE 50 but the FOR/NEXT loop as used here is a short timing loop that does the same thing only it is a smoother transition instead of the jerky PAUSE.

EXPANDING RAM

- by Jack Paget

The article on the right appeared in Gladstone's catalogue. It seemed quite simple, and as I had found the 1K RAM small, I thought 2K would allow more flexibility during my learning ZX81 programming and logically to me postpone a decision of what size RAM to buy.

ZX-81 TECHNICAL CORNER
By Brett Hilder

1) Question: Can I get more memory inside my unexpanded ZX-81?

Answer: A 2K RAM chip can be added internally to provide 2K internal RAM for the AZ81. The ram IC4-4118 should be removed and one 6116-3 should replace it. Link #2 should replace Link #1. (Please note that internal RAM is always disabled when additional memory links are used).

*ANY internal modifications to a ZX-81 will void the warranty and should be attempted by individuals who have electronic expertise.

Upon removing the computer circuit board, I found mine did not look like the one pictured on page 162 of the AX81 Manual.

All IC's were soldered to the PC Board. There were (2) two 2116 chips side by side labelled Ic4a and Ic4b.

I removed Ic4a, using solder wick on each of the pins of the Ic4a. Then, using Radio Shack's Spring Extractor, I heated the pins simultaneously, using Ungars Ic hearing pad on a 42 watt soldering pen. The Ic popped off the board. Apparently it would be wise to use a 3-wire grounded tip iron.

I then, using a sharpened toothpick, and the computer clamped on a vice, pushed the sharp end into each of the Ic pin holes, having melted the solder from the back of the PC board. Once each hole was cleared, I soldered a socket for the 6116 Ic into place. Then I inserted the 6116, checking carefully that the chip's #1 pin engaged the circuit's #1 trace.

continued........

I then carefully cut (using a dentist's pic) the +5V trace to Ic4b. This is not noted in the article but Dennis mentioned this should be done.

After assembly hooking up power and TV, I found I had a white screen, which proved to be a permanent glitch, no cursar, (K,L,G,F,) and when switching power on and off, I would sometimes get a black screen.

Needless to say, the 90 day warranty was now invalid. The author, Brett Hilder was puzzled. The distributor suggested $35.00 minimum to $129.00 maximum would fix it. Not what I wanted.

After 2-3 weeks of "asking people what I should try next" and reading about how Ic circuits work and checking the circuit board for breaks or shorts in the traces, I finally pulled the 2nd Ic4 off the board, reconnected the computer to the TV and it worked.

On the "HeatLoad" program I was working on, I was out of RAM and after tidying up the display, I found I had used over 1600 bytes.

Needless to say, I'm quite happy and I am quite likely to get into the circuit again, to add "Push to Reset" and possibly a composite video plus sync circuit to connect to a monitor.

==